RC-6326 (#27179) 12/14/76
Computer Science 49 pages

AD A043033

Research Report

BEHAVIORAL ISSUES IN THE USE OF INTERACTIVE SYSTEMS

Lance A. Miller and John C. Thomas, Jr.

IBM Research Laboratory
Yorktown Heights, New York 10598

14 Dec 76 | 33p.

D

This research was supported in part by the Engineering Psychology Programs, Office of Naval Research, Contract Number N00014-72-C-0419 Work Unit Number NR-197-020.

IBM Research Division
San Jose · Yorktown · Zurich

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RC6326	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Behavioral Issues in the Use of Interactive Systems		5. TYPE OF REPORT & PERIOD COVERED: Interim Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Lance A. Miller* John C. Thomas		8. CONTRACT OR GRANT NUMBER(s): N00014-72-C-0419
9. PERFORMING ORGANIZATION NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR-197-020
11. CONTROLLING OFFICE NAME AND ADDRESS: International Business Machines T. J. Watson Research Center, P.O. Box 218 Yorktown Heights, New York 10598		12. REPORT DATE: 12/14/76
		13. NUMBER OF PAGES: 49
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research Code 455 Arlington, Virginia		15. SECURITY CLASS. (of this report): Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

man-computer interface displays editors human factors interactive systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ABSTRACT: This paper identifies behavioral issues related to the use of interactive computers primarily by persons who are not computer professionals, so-called 'general users.' This is not an exhaustive literature survey but instead provides: (1) a structure for discussing issues of interactive computing, and (2) the authors' best estimate of importance behavioral problems, with suggestions for solutions.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The discussion is limited in this paper to general issues which do not take into account the user's particular task. The two major topics are System Characteristics (performance, facilities, and on-line information), and Interface Characteristics (dialogue style, displays and graphics, other input/output media).

RC 6326 (#27179) 12/14/76
Computer Science 49 pages

BEHAVIORAL ISSUES IN THE USE OF INTERACTIVE SYSTEMS:

Lance A. Miller and John C. Thomas, Jr.*,

IBM Research Laboratory
Yorktown Heights, New York 10598

ABSTRACT: This paper identifies behavioral issues related to the use of interactive computers primarily by persons who are not computer professionals, so-called 'general users.' This is not an exhaustive literature survey but instead provides: (1) a structure for discussing issues of interactive computing, and (2) the authors' best estimate of important behavioral problems, with suggestions for solutions.

The discussion is limited in this paper to general issues which do not take into account the user's particular task. The two major topics are *System Characteristics* (performance, facilities, and on-line information), and *Interface Characteristics* (dialogue style, displays and graphics, other input/output media).

* Some of the authors' work was supported, in part, by Engineering Psychology Programs, Office of Naval Research.

* We gratefully acknowledge the suggestions of Pat Goldberg, Ashok Malhotra, and Phyllis Riesner who commented on an earlier version of this manuscript.

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication elsewhere and has been issued as a Research Report for early dissemination of its contents. As a courtesy to the intended publisher, it should not be widely distributed until after the date of outside publication.

TABLE OF CONTENTS

INTRODUCTION

We are concerned, in this paper, with behavioral issues in achieving effective computer environments for man-computer interaction. Our target user is primarily someone who is *not* a computer professional (e.g., programmer or systems engineer), but who interacts with computer systems to achieve specific task goals of any nature -- what one might designate as *general users*. We believe that the key to achieving substantial improvement in the computing environment lies in a better understanding of a general user's task characteristics and goals. In particular, we identify two classes of human activities, each of which has different implications for providing an improved user-computer environment. There is, first, the class of *routine tasks*, which can be characterized as being governed by some specific procedure, the activity cycle being initiated by some external event (i.e., event-driven), which cycle is of short duration and produces, typically, information-conserving (or simple information-reduction) transformations on the input data. The second activity class is that of problem-solving, and a number of types can be identified, each with different requirements and objectives (Miller & Thomas, in preparation). Each of these classes, and the specialized tasks within them, imply particular and different functional capability desirable for the most effective performance.

Despite this orientation, our approach here is to consider the behavioral issues that exist even when *nothing* is known about the particular user or the user's tasks of the moment. Such a collection of considerations provides a reference that *always* applies, regardless of the user's activity. We provide a detailed and (hopefully) comprehensive taxonomy of features that characterize interactive systems. For each feature, we consider some alternative implementations, emphasizing the *functional capability* that should be provided in the interactive environment -- both those specific functions which the user should be able to command of the system, and also those support activities which the system should automatically provide for the

user. Literature citations were chosen to provide representative -- but not exhaustive - sampling of relevant studies.

To give some substance to, or baseline reference for, our recommendations, the reader should assume that the general user is communicating with a powerful computer system via some keyboard and display device. The functional capability recommended is to be understood as additional to such a system. More concretely, the following computer system (the one with which we are most familiar) may be assumed: the Conversational Monitor System (CMS) component of IBM's *Virtual Machine Facility,* VM/370, as installed on a 370/168 computer using an IBM 3277 alphanumeric display.

The following reviews are *useful introductions* to the indicated areas. For *general human factors engineering* applicable to interactive systems, the reader is referred to the following: Martin, T. 1973; McCormick, 1970; Meadow, 1970; Meister & Rabideau, 1965; Singleton, 1974; Van Cott & Kinkade, 1972. For reviews of work concerned with *user-computer interfaces* see: Bennett, 1972; Davis, 1966; Kemeny, 1972; Licklider, 1968; Martin, J., 1973; Martin, T., 1973; Martin, Carlisle, and Treu, 1973; Palme, 1975; Rouse, 1975; Shackel, 1969; Tomeski and Lazarus, 1975; Walker, 1971. For a *conceptual analysis of users' tasks,* and an excellent analysis of the *'ease of use' question,* see R. B. Miller, 1969, 1971, respectively.

1. SYSTEM CHARACTERISTICS

1.1 PERFORMANCE

1.1.1 Batch vs. Time-Sharing

This is much less of an issue now than it was a few years ago when interactive systems were just becoming available. One of the most interesting findings to come out of this (sometimes crude) batch vs. time-sharing work (aside from the slight superiority of interactive systems) was that individual differences among programmers were often a much larger and important

source of performance variability than system differences (see, e.g., Adams & Cohen, 1969; Sackman, 1970; Weinberg, 1971). Current discussions typically assume interactive systems and focus on *what kind* of interactive systems are best. This paper concentrates on this question.

1.1.2 System Response Time

The definition of system response time depends on the nature of the computer system involved. In most studies input to the system was via a keyboard, which keyboard was 'locked' (preventing the user from issuing further inputs) until some action (completion or error message) was taken by the system on the input. System response time -- SRT -- thus most oft[illegible] to the time the user is prevented from doing further work following a single [illegible] input until the disposal of *that* input. (Note that present-day displays most often do *not* lock out the user but permit multiple commands to be issued).

There is some data showing the effects of increasing SRT -- e.g., on subsequently increased *user* response times (Boies, 1974), and on (predicted) decreasing user acceptability ratings (Carbonell, Elkind, & Nickerson, 1968). Lancaster and Fayen (1973) review similar considerations of system response time from the point of view of information retrieval. R.B. Miller (1968) provides perhaps the best conceptual analysis, giving suggestions for maximum SRTs as a function of 17 different kinds of user input (e.g., light pen entries, request for next page). He and others (e.g., Engel & Granda, 1975; Schwartz, 1969) make the cogent point that the impact on users of longer SRTs depends upon the complexity of the task engaged in -- where, for more complex tasks, longer SRTs might, in fact, be helpful. Boehm et al carried this idea a little further and explored the effects, with inconclusive results, of 'locking out' the user for variable periods *after* the system had responded -- so as to induce the user to concentrate more on the immediate problem at hand (Boehm, B.W., Seven, M.J., & Watson, R.A., 1971). Sackman and Gold (1968) also endorse such a notion.

The profound technological improvements (hardware *and* software) that have taken place since the above studies were conducted permit *greatly improved system response times*, and concerns about the *absolute magnitude* of delays may no longer be warranted. However, as Carbonell *et al.* (1968) and others have pointed out, it is the *variability* of delays, not their magnitude, which is often the most distressing factor to users. As for the Boehm *et al.* 'lockout' proposal, or other artificial restrictions on computer function, such presumptive actions may run the risk of engendering significant (and disrupting) user dissatisfactions.

1.1.3 Availability and Reliability

While there may not have been any studies specifically investigating the effects of poor availability or reliability on user performance, the importance of these factors is high on most people's list of important system criteria. One important, but implicit, principle appears to be: Users will be unhappy with *any* system degradation, no matter how good the normal performance. For many computer applications -- e.g., dispatching, combat information systems, process control -almost no loss in availability or performance errors can be tolerated; there is, therefore, little point in studying behavior under known suboptimal conditions (except, perhaps, to determine how a *particular* working system might best limp along with forced limits on availability or reliability).

1.2 FACILITIES

1.2.1 System Command Languages

A user's access to a computer system and its various facilities is, in almost all cases, via a system command language. Probably no other feature is more important in determining an individual's effectiveness in using a computer system than this aspect. The user is often placed in the position of an absolute master over an awesomely powerful slave, who speaks a strange and painfully awkward tongue, whose obedience is immediate and complete but woefully

thoughtless, without regard to the potential destruction of its master's things, rigid to the point of being psychotic, lacking sense, memory, compassion and -- worst of all -- obvious consistency.

There are a diversity of system command languages just as there are of programming languages (e.g., Sammet, 1969), and considerable effort has been expended in the advocacy and intuitive evaluation of their relative merits (see, e.g., Unger, 1975). While there has always been strong sentiment for modelling command (and programming) languages on natural languages, little progress has been made (but see Heidorn, 1976; Kelly, 1975; Miller, L., 1976; also, for a thorough account of the complexities of natural language commands, see Rescher, 1966). It is very likely that command languages will continue for quite a while as, at best, pidgin dialects.

There are several important behavioral issues which *can* be addressed, however. A typical system command language is assumed in the discussion, wherein the first part of any command is the reserved command word, followed by one or more 'words' -*arguments* -- which specify various options or alternatives for realizing the particular command; the arrangement of the arguments is the *argument format.* When the user fails to specify certain arguments, the computer system may automatically assign *default* values to these. An example of a command-argument string is:

PRINT TEXT MIXED CENTER

(cmnd) (a r g u m e n t s)

where TEXT is the data set to be PRINTed, MIXED specifies upper and lower case letters, and CENTER causes the listing to be centered on the page. While a data-set name would always be required, the *defaults* might be to use uppercase letters and left-margin printing.

1.2.1.1 Command Organization

One issue concerning command languages is the manner in which the system commands are organized. Boies (1974) found that a majority of users on a large time-sharing system used only a very small number of the numerous system commands available -- often employing the simplest, least powerful, form of such commands. Such findings could result from organizations of the commands which made it difficult for users to recall them when they were needed (also see Kennedy, 1974). In an unpublished study, Boies subsequently compared two types of organizational approaches: a small number of broad generic commands with a large number of arguments vs. a large number of quite specific commands with very few arguments. The generic organization appeared to be the more useful. However, much more work needs to be done before one could justifiably propose an 'optimal' organization strategy.

1.2.1.2 Argument Formats

There are at least two distinct methods of formatting the arguments for commands: specific kinds of information may be assigned in a fixed relative or absolute position in the argument string -*positional format* ; alternatively, arguments may be given as permutable strings of special words indicating the argument type as well as its value -*keyword format*. Both types of formats (and some deadly combinations) occur in real systems. Concerning performance predictions for these formats, it would seem that the positional format would impose the greater memory load for the user, since the values of the arguments must be remembered in both cases, and remembering the position is an additional burden. There is some support for this view from an informal experiment in which error-rates were found to be much higher for positional formatting (cf. Weinberg, 1971).

1.2.1.3 Prompting and Defaults Within Commands

There remains the issue of what to do when the user, for one reason or another, fails to specify some information that either could or should have been provided. In the narrow sense this is the situation where a command has been issued, but one or more of its arguments has not been given. There is, first, the option of disregarding the command entirely -- and many systems so do -- and, second, if not disregarded, what to do about the missing information.

There are several options for prompting the user for missing information, ranging from a brief listing of the missing argument names to a full display of potential values for each of the missing items along with an easy means of indicating one's choice. For a relatively small set of alternatives, permitting users to choose from among them might be an easier task than asking users to generate the alternative; however, the selection mode becomes much less desirable when the display of alternatives is very complicated or takes a long time to produce.

As an alternative to prompting, it may be possible to assign a value -- a *default* value -- automatically to some of the missing arguments. This is one of the most powerful of existing computer system concepts for achieving a *user-oriented* environment. Essentially, the use of defaults constitutes an agreement between user and computer as to what a 'normal' or 'usual' working environment might be. However, problems can arise, e.g., the user does not know of the default assignment system or doesn't understand the defaults, or the user does not have a convenient means for changing the defaults. Perhaps the computer should (optionally) display assumed defaults to the user.

There is considerably more that can be done to extend the default concept. For example, users might have separate profiles of defaults which are appropriate to different tasks, such as editing, programming, etc. Various alternatives for prompting and defaults have been considered (e.g., Martin, J., 1973), but, again, there has been very little direct empirical assessment.

1.2.1.4 Prompting and Defaults Between Commands

One of the ways in which the default concept can be more broadly implemented is to extend it to strings of commands, rather than just to strings of arguments within commands. The idea is that the computer system would be given expectations concerning certain 'normal' *sequences* of user commands. Such expectations could provide the basis for the following kinds of 'intelligent' inferences by the computer system: (1) supplying missing command(s) in a certain sequence of commands; (2) supplying missing arguments for a command on the basis of arguments supplied to previous commands; (3) recognizing that, for a series of inputs of arguments without commands, a prior-given command should be supplied. This idea is not to be confused with the concept of 'macros' or specialized functions written for particular circumstances. The suggestion here is that this property of providing for defaults *between* commands is a characteristic of the host computer's operating system, not a special-purpose user program.

1.2.2 Editing

1.2.2.1 Usage

In an analysis of the commands actually issued interactively to an IBM TSS/360 system, 75 percent were found to be editing commands (Doherty, Thompson, & Boies, 1972; also see Boies, 1974). This extremely high editing usage did vary as a function of the type of user: programmers issued a not-so-low 50 percent editing commands while users preparing text documents issued over 80 percent. Taking usage as a measure of importance, editing facilities thus appear to be *the most important facility provided* by computer systems. Accordingly, detailed attention will be given to these issues. (For a survey of on-line editing systems, see Van Dam and Rice, 1971).

1.2.2.2 Computer Program Editors

For the most highly-used programming languages (e.g., FORTRAN, COBOL) there are *three* characteristics which should influence the nature of an editor for preparing programs: (1) programs are primarily composed of fixed-length records (typically 80 characters); (2) information is *not* usually broken across record boundaries -- so that each record, or line, constitutes an independent entity (ignoring comments); and (3) there are fixed fields within records of one or more character positions which are reserved uniquely for various aspects of the programming language (as labels begin in column 1, commands begin in column 7, etc., in FORTRAN).

These characteristics suggest that, in addition to possessing the usual character/string editing capabilities (e.g., change, insert, delete), well-designed program editors should: (1) be oriented towards dealing with individual lines within a data-set; (2) have extensive provisions for establishing fields and moving from field to field (e.g., via tab controls); (3) provide for easy entry of full-length records of characters (such as '*') to serve as delineators of parts of the program; (4) have special, easy-to-use, commands for moving groups of one or *more lines* from one position to another, or for copying blocks of lines from one program data-set to another; (5) provide a scheme for numbering lines to permit communication between processors of the program (e.g., the compiler -- 'ERROR IN LINE 43 ... '), as well as for local line-oriented editing. (6) In addition, there are a variety of features that may be useful for particular languages (e.g., checking for parenthesis balancing in an interactive LISP envionrment).

Most program editors do, in fact, have these features in one form or another (e.g., the Quick EDitor, Deutsch & Lampson, 1967; the CMS editor, IBM, 1976). Alternative variants of these features have not been evaluated, however; nor have functional requirements for program editing been studied empirically.

There is an additional aspect to computer programming, which, possibly, could be taken advantage of in editors: the lexicon of program commands, values, and operators is usually quite small, and the arrangement of command arguments is usually in positional format. Thus, special keyboards or abbreviations, etc., might be provided for selecting commands, and the editor could have certain prompting/default modes for obtaining the command arguments. There is some discussion concerning the former (e.g., Martin, 1973; Meadow, 1970; Walker, Gurd, & Drawneek, 1975), but not the latter idea.

1.2.2.3 Text Editors

A second class of editors is one designed for the preparation of text materials -- reports, memoranda, manuals, etc. As with programs, there are several characteristics of text which have implications for appropriate editing features: (1) the text is (almost) never input or initially formatted in any absolute way -- i.e., there are no fixed-lengths to segments of input documents, nor, within any segment, are there special 'fields' reserved for particular types of information; finally, there are few instances of text segments corresponding to fixed-length or positionally formatted argument strings; (2) there *are*, however, two universal units of segmentation within text -*words* and *sentences* ; paragraphs and larger kinds of segmentations are also found quite frequently; (3) there are typically rather a large number of different word (and phrase) types in a document; however, some types have a much higher token frequency than others; there may also be many synonymity relationships among types; (4) in contrast to computer programs, text materials are formatted for final output in a huge variety of ways, with many possibilities for, e.g., character-font, indentation, and spacing variations.

These characteristics imply that a well-designed text editor should have at least the following functional capabilities: (1) text input to the system should be represented (at least to the user) as one long character string; easy means should be provided for adding, inserting, or deleting text segments with respect to this string; (2) location of targets within the text data-set should

be on the basis of *words*, or word-phrases, in addition to a simple character-string search capability; (3) it should also be possible to search the text for *synonyms* of the desired target; (4) functions should exist for easily moving or copying text portions on the basis of sentence, paragraph, or higher-order segments; (5) facilities should be provided to abbreviate, or otherwise code, long input strings into much shorter strings (as, for example, 'IBM' could be typed to input the phrase 'International Business Machines, Inc.'); (6) means should be provided for the user to indicate all specifics as to the manner in which the final output should be formatted -- including indentations, font selection, margins, line-lengths, headings, etc.

The situation for text editors is much worse than that for program editors. In most cases, text editors do not possess the above characteristics and are simply line-oriented program editors with many of the program-editing features turned off (e.g., tabs, automatic advance to next record after fixed-length input, etc.). Text is stored as line-records, displayed to the user as such, and, worst of all, searched on this basis. Specific problems which severely limit the utility of text editors are discussed in detail below.

(1). Problem with searching on string basis -- In text preparation the user is almost always thinking in terms of *words*, and higher units composed of words; the exception is when mis-spellings are being considered, a very different cognitive situation. Suppose a user wished to change statement (1) into (2):

(1) I remember what he said to me.

(2) I remember what he said to you.

Within most text editors, it would seem reasonable to use the CHANGE command, as 'CHANGE/me/you/'. Unfortunately, this would produce (3):

(3) I reyoumber what he said to me.

It is true that with concentrated attention (and four extra key strokes) the user could anticipate the problems and try to indicate the 'wordness' of the desired targets by surrounding them with blanks, as:

'CHANGE/ me / you /'.

However, this would fail, since the string ' me ' does not occur in (1). The change command would have to be e.g., 'CHANGE/ me./ you./'.

This character-string orientation of text editors is a most inconsiderate one for word-processing users who may well have to rewrite these editors for production-environment use (e.g., Stone & Webster Corporation, 1973).

(2). Problem with line-oriented representation -- Consider the following excerpt from a computer-resident document:

...

References to the visitations of Quir

aliens are recorded in the writings of Alimar

Shazam from the fifth century, B.C., and also in

...

A researcher searching for references to Alimar Shazam might very reasonably issue the command: 'LOCATE/Alimar Shazam/'. The editor would, alas, inform the user that there was no mention anywhere of such an entity: the line-oriented search would fail since the target word-phrase is broken *between* lines of the document. Clearly, the search should be made as if the text were one long string of words.

A second problem with the line-oriented representation is that it is a *line,* not a sentence , which is returned by the search functions. A user requesting references to Quirs would be returned only the sentence fragment. This is perhaps tolerable if the user is continuously interacting with the system and can read the surrounding context; however, for automatic

processing, e.g., collecting all sentences with certain features, such a procedure would be unacceptable.

It should be noted that program editors cannot be made to search on a sentence basis simply by searching for a character string in the context of a period or semi-colon. These punctuations also have other separator functions (e.g., as in 'J. Doe' or 'Abel, 1956; Baker, 1957'). Separate sentence (and paragraph, etc.) markers would have to be defined.

(3). Problem with multiple target -- Now suppose the user wishes to search for any (or only) one of a set of targets -- e.g., 'aliens', 'Quirs', or 'fifth century'. Most editors do not have simple and direct means for expressing such disjunctive search requests.

(4) Problem with KWIC search requests -- Very few editors would further be able to service directly KWIC (Key Word In Context) search requests, wherein a possibly complex target (e.g., of multiple, not necessarily adjacent, words) is to be searched for within some text unit (sentence, paragraph) subject to the presence or absence of some other target. An example might be: locate all sentences in which the phrases 'alien' and 'Alimar Shazam' both occur, but the phrase 'sixth century' is absent.

The above discussion relates to the group of 1-4 desiderata of text editors. With respect to the fifth, functions for assisting text input, based on an abbreviation strategy, are not yet available. However, such a capability -- termed 'short-type' -- has been behaviorally evaluated in the laboratory, and there appears to be considerable utility to this approach (Schoonard & Boies, 1973).

In contrast to the above points, concerning the sixth point of output formatting, many editors possess excellent facilities for: (1) formatting textual materials into particular document formats (e.g., Farjman & Borgelt, 1973; IBM, 1975) , and (2) for converting such documents into user-defined formats appropriate for book-quality printing (e.g., IBM, 1973). Much more

could be done, however. In particular, the user could be permitted to select from a library of document types or publication styles. The selection would determine not only simple spacing, margin, etc. format values, but would also indicate more complex features such as how the structural headings of a paper were to be composed (e.g., in terms of centering, fonts, capitalization, etc.) or how different sections of a paper might be formatted (e.g., the abstract centered and single-spaced vs. double-spacing and longer line-length for the text body). The user's selection could also determine what additional information -- obtained from data on file or by prompting the user -should be inserted into the document (e.g., distribution lists and user identification/location information).

1.2.2.4 Structural Editors and The Future

In the discussion so far it has been presumed that the target to be located, changed, displayed, etc., has been specified literally by the user. This may be very inconvenient if the user is interested only in dealing with *any* instance of a class of things: the user would have to specify literally every member of the class, and for large classes this could be quite time-consuming to do so. It would be much more convenient if the user could define some 'structure' to be imposed on the literal characters or words of the text (these latter corresponding to 'terminal symbols' in formal grammars). The structure would define how the lexical items were to be grouped into equivalence classes, subsumed under various concepts or relations, and how these concepts were further grouped into higher-order concepts, and so forth (i.e., the 'non-terminal symbols'). The user could then specify editing commands on the basis of these higher-order classes by issuing the appropriate non-terminal character string. In terms of the example in the previous section, a user could have defined a structure which would hit on 'fifth century' in response to a request for references to *historical dates*. Such an approach is similar to concepts of affording users systems which take into account, in various ways, the *semantics* of the user's domain (e.g., Burton, 1974; Skinner, 1972; Tarnawsky, 1972; Wilks, 1973).

This concept of a structured editor may seem similar to certain information retrieval systems, wherein the user can request information for various classes of information. In these latter systems, however, the data-base is not the unformatted text document presumed here, but an already structured representation in which classes of information are mapped onto specific fields of separate information records. In the present concept, a user would define a personal structure to be superimposed over an unformatted, unorganized linear word record. In this sense a structured editor is related to the concept of a *personalized information system*, in which the descriptors or indices of documents are generated and organized according to the personal weltanshauung of the user (e.g., Mittman & Borman, 1975; Sauvain, 1971).

There are, of course, a number of problems in implementing and making usable such a system, not least of which is the problem of specifying the user's structure (equivalent to writing a very complicated formal grammar). Nevertheless, experimental work has begun with such editors oriented towards assisting programmers in their work (e.g., Donzeau-Gouge, 1975; Kruskal, 1976). The Kruskal work, in particular, with its innovative coding of information by color and flashing, has potential applicability in several other areas, including design as well as text-processing.

One final approach, which warrants attention for its far-reaching implications, is the concept of editing solely by telephone. A user would initially dictate a document, using a touch-tone telephone connected to a remote computer and would indicate sentences and paragraphs in the audio document by special button-push combinations. Subsequently, the document could be retrieved and the touch-tone buttons used to play-back and edit this audio file. Such a system is presently being researched under the management of S. Boies at the IBM Watson Research Center.

1.2.3 File Manipulation

1.2.3.1 File Manipulation Commands

The editing usage data (Boies, 1974) indicates that users usually created a number of different files -- both texts and programs. There are a number of manipulations that will be performed on these files: displays, sorts, merges, embeddings, and hard-copy output -- on local terminals or on remote line-printers, or via punched cards, paper or magnetic tapes. Simple, direct, and consistent means should be provided for accomplishing all of these operations independent of the nature of individual files -- their content, size, or structure. Further, such means should be constant across all of the environments in which the operations are permissible -- e.g., from within editors, under control of programming facilities, or at the highest level. While few would argue with such desiderata, there are many systems in existence which violate these principles.

1.2.3.2 Querying the File Catalogue

In addition to some *filename*, files may also be characterized as to *filetype* -- the kind of file, e.g., FORTRAN, text -- and *filemode* -- a reference to a particular (real or virtual) storage location of the file (these particular terms are used on the CMS VM/370 system; cf. IBM, 1976).

The user should have an easy means of determining what files are possessed, in toto, or as a function of some aspect of their filename (e.g., all files with a name beginning with 'J'), filetype (e.g., all PL/I program files), or filemode (e.g., all files located in archival storage). Further, the user should be able to produce any kind of sorted listings or displays of the complete file catalogue. Finally, the user should be able to aggregate any sorted collection of files into a new file to permit block transfer or other manipulation of the file group. Although

file querying and sorting capabilities are typically found in most systems, the characteristics or problems of usage are unknown.

1.2.3.3 Querying Formatted Data Bases

For many applications (e.g., airlines reservation systems, management information systems) the user needs to retrieve specific data from formatted files. Many query languages have been proposed for such purposes (Sammet, 1969),. geared mainly toward dedicated users within specific applications. There have been some attempts, however, to provide query languages for the casual user. The proposed user interfaces for such systems include formal tabular interfaces (Zloof, 1974, 1975), graphic languages (McDonald, 1975a, 1975b), structured sequences of English-like commands (Chamberlin and Boyce, 1974) and natural language (Codd, 1974). Behavioral work has shown that non-programmers could learn to use a laboratory query language in about 3 hours (Thomas and Gould, 1974). An extension of this system, *Query By Example,* also provides for file query commands in a syntax consistent with the data query commands. A further review of various options for query systems is provided by Martin, (1973). The reader is also referred to behavioral studies on the potentially relevant topics of (1) presenting tabular information (Wright and Fox, 1970), (2) the difficulties people have with quantifiers (Thomas, 1976a), and (3) people's strategies in asking a sequence of queries to solve a problem (Malhotra, 1975, Thomas, 1976b).

1.2.4 Data Manipulation

The mushrooming popularity of the interactive programming language *APL* might be due in no small way to the ease with which users can perform a variety of immediate computations. (e.g., Pratt, 1975). Only exceptionally do such facilities exist outside of an APL environment, however. Nevertheless such a capability is, highly desired in *all* systems to permit users to display, manipulate, transform, evaluate, and format their numerical data files. Many addition-

al specialized display and transformation functions could be provided for specialized uses, such as for applications involving geographical data bases (e.g., Carlson, 1975; Grace, 1975), for management decision support systems (e.g., Bennett, 1975; Gorry & Morton, 1971; Newsted & Wynne, 1976), or for specific scientific disciplines, such as Behavioral Sciences (e.g., Yntema, 1974). Despite such activity, there are almost no empirical (or theoretical) analyses of what people do when there are manipulating data in the interest of solving some problem. Such information is clearly mandatory for efforts designed to provide computer support of such behavior.

1.2.5 Programming Language Support

We consider here only the overall language processing capabilities that could be made available on interactive systems, and we make no evaluations of or comparisons among specific languages or specific processors. The reader interested in such topics is referred to the following, partial list of sources: general discussion of programming languages (Elson, 1973; Pratt, 1975; Sammet, 1969), programming tools and management (Aron, 1974; Metzger, 1973; Miller, 1973), comparison of programming languages or computer systems (Reaser, Priesman, & Gill, 1974; Sackman, 1970; Sime & Green, 1974; Sime, Green, & Guest, 1973; Schwartz, 1969), discussion of new application-oriented languages (Goldberg, 1975; Hammer, Howe, Kruskal, and Wladawsky, 1975), and behavioral problems in programming (Miller, L., 1973, 1974, 1975; Weinberg, 1971).

1.2.5.1 Programming Language Processors

As with file manipulations, the facilities for executing and otherwise processing computer programs should be simple and consistent across programming languages and across processing facility. Requests for special features (e.g., *producing cross-reference* lists) could be accomplished by using optional arguments in the invocation command (cf. section on System

Command Language). Alternatively, since users tend to program in only one language (cf. Boies, 1974), some provision could be made to define a profile of the individual user' preferences in program processing -e.g., personal choice of output devices, special features, syntax-checking options, etc. The defaults for invoking the language processors could then be matched to this profile.

1.2.5.2 Debugging and Testing Facilities

Data concerning programming errors associated with *batch* systems (e.g., Nagy & Pennebaker, 1971; Sackman, 1970) is more available than for interactive systems. In one study of a time-sharing system, however, a rather amazing 80 percent (or more) of the submitted programs were found to have *no* syntactic errors (errors detected by the compiler; Boies & Gould, 1974). In another, similar, study, Moulton and Muller (1967) found that 66 percent of college student users' programs compiled perfectly. It thus appears that, whatever problems users may have in programming, the primary difficulty is *not* in producing programs with correct syntax. Thus, investment in more and better specialized syntax-checking facilities -- e.g., line-by-line syntax checking for FORTRAN -- not only may not be justified, but also the facilities might not even be used (cf. Boies, (1974), for evidence that this is true on a current system).

Users *do* modify syntactically correct programs (e.g., Boies, 1974), however, when the programs do not perform as they were intended to. That is, the programs are perfectly grammatical, but the algorithm for processing the information is incorrect in some *conceptual* respect (these errors may be very common -- over 40 percent in one study; Walter & Wallace, 1967). In addition to the problem of correcting such *conceptual* errors, apparently correct programs must be tested for correct handling of *all* input by excercising the programs over a wide range of parameter-values and input. There is thus a vital need for computer systems to provide for users suitable debugging and testing facilities.

The requirements for such facilities, and the efficacy of specific tools, has been extensively considered (e.g., Balzer, 1969; Boies & Spiegel, 1973; Brown & Sampson, 1973; Gaines, 1969; Rustin, 1971; Satterthwaite, 1972). Many systems provide generalized and extensive tracing and monitoring capabilities (like selective/full on-line/post-mortem traces, control-path monitoring, etc; see Aron, 1974; Metzger, 1973; Van Tassel, 1974); some programming languages, as opposed to the operating systems, even provide sophisticated debugging options (e.g., ALGOL W, cf. Van Tassel, 1974 pp. 151-152).

There are a number of packages for generating test cases to test programs (e.g., Van Tassel, 1974), but these will only be able to test for some general set of possible invalidities. Testing packages could be improved, for example, if the programmer or designer specified ranges, bounds, exceptions, etc. on the program variables: this information could be embedded in the program, and test-cases could be generated using these restriction specifications. Another, very promising, new approach is to treat the program much as a set of algebraic equations and supply symbolic inputs to permit partial equations and then execute the program *symbolically* (e.g. Hantler & King, 1976; King, 1974). Such an approach would permit testing for whole classes of inputs at one time, and, with suitable interactive direction from the programmer, greatly increase the comprehensiveness of testing. Between these two suggestions for improving program testing is the concept of automatically analyzing programs to determine the ranges or other restrictions on variables assumed at various points in the program; such deductions could then be used to guide and restrict the selection of test cases (e.g., Harrison, 1975).

One area in which there could be substantial improvement concerns facilities for *modifying* programs. We suspect that the programmer intent on modifying the program will often be concerned with the flow, use, and modification of *data*. While there is significant work on data-flow analyses (e.g., Allen & Cocke, 1975; Rosen,1975), and much progress in automatically providing all static information about data entities (e.g., Tapscott, 1974), there is almost

nothing known about the *behavioral* problems of program modification or what tools might be useful.

1.2.6 Inter-User Communication Capabilities

1.2.6.1 Shipments of Data Files

Ignoring costs, it would be desirable to allow users to transfer data files among themselves, either on the same system or between different systems connected by phone-lines. Ideally, the shipment -- and reading -- commands should be no more complex than:

SHIP <USERNAME> <FILE> <COMPUTER SYSTEM>

READ SHIPMENT <NEW FILE NAME>.

As with all file manipulation commands, the user should be able to transfer files in the same way regardless of the size, content, or structure of the file.

1.2.6.2 Real-Time Interactions

One of the most exciting potentials of computers is that of distributed interactive work. The scenario is that of widely-scattered users -- possibly in different countries -working together on a common task, e.g., software development via computer display terminals. One can imagine several programmers discussing a particular program, displayed to all, with provisions for each person to selectively point at or annotate aspects of the program, with further capability for executing and tracing the program from each station. Englebart has long been one of the strongest advocates of such a scenario, particularly emphasizing text preparation (e.g., Englebart, 1962; Englebart, Watson, & Norton, 1973), but the idea is present in many other forms, particularly under the notion of video teleconferencing (see Dunn, 1975).

One problem that may occur with work shared work via video-only terminals is that the audio channel seems very important, at least for two-person problem solving (Chapanis, 1971) Using computer displays for distributed work presents issues concerned with the *communication*

channel, and issues concerned with the *work channel.* The communication channel refers to various messages that workers may wish to send to each other which should not be included in the final text, program, etc. The work channel refers to the evolving end product of their shared activity; that is, the program, text, system etc. With respect to communications, some of the problems (and example solutions) are: (1) providing for a *separate display area* for the user messages (e.g., partition the screen into upper and lower parts); (2) managing the interchange -- preventing two or more from 'talking at the same time', buffering the input from the next 'speaker', etc. (e.g., users must transmit a signal corresponding to raising one's hand for recognition, and transmission of their message is blocked by a *central controller* until such a message is received by the controller and acknowledged); (3) identifying the participants (e.g., users enter identification name which the system automatically inserts at the beginning of each transmitted message); (4) recording the communications and accessing the communication record (e.g., write a record into a historical file for each message; temporarily replace the working display area with the communications file when necessary).

Potential problems with the work channel (and possible remedies) are: (1) providing each user the capability to 'point' selectively at the work (e.g., provide multiple cursors for each, *separably controllable*); (2) managing the modifications of the work -- how is editing to be done, how do individuals obtain personal copies of work, etc. (e.g., users sign-on under their own ID but link to a common ID; this common working environment has provisions for saving work, editing, etc., while each user continues to have access to personal files).

These and other problems have been addressed to some extent in Englebart's work (e.g., Englebart et al, 1973). It is somewhat difficult to do much more than speculate at this point since there is very little experience with such situations, and no theory of human communication is able at the present to predict the problem areas. There is some encouraging ongoing work, however: Chapanis' controlled studies of communication under a variety of communica-

tion modes (Chapanis, 1971; Ochsman & Chapanis, 1974), the development of much more sophisticated models of knowledge (e.g., Bobrow & Collins, 1975; Mann, 1975) and the development of methodologies for empirically studying natural multiple-person inter-communications ('multilogues'?; e.g., Mann, Moore, Levin, & Carlisle, 1975).

1.2.7 Recovery Philosophy

Recovery refers to the re-instatement of some past state of the computer system environment, usually after some kind of error or malfunction. There has been almost no general consideration of recovery approaches (see Engel & Granda, 1975), but four types of recovery situations, with respect to the user, can be identified: (1) user correction of data input, (2) user abolishment of prior commands, (3) abnormal exit from within some program or programming language processor, and (4) system crash.

With respect to (1), user corrections of input data, most displays now provide for local editing (i.e., correction) before transmission. Following transmission, the user should have available some reserved key or command to provide for the correction, without having to invoke an editor. It seems likely that data input errors are usually caught immediately by users, and extensive buffering of input data should not be required.

A more complex situation, however, occurs for situation (2), when a user wishes to 'undo' the effects of some number of prior commands -- as, for example, when a user inadvertently deletes *all* personal files. Recovery from such situations is handled by most systems by providing 'backup' copies of (all) users' files, from which a user can get restored the personal files as they were some days previous. While this is perhaps acceptable for catastrophic errors, it would be quite useful to permit users to 'take back' at least the immediately preceding command (by issuing some special 'undo' command). Implementation of such a feature would require buffer storage of *all* of a user's files which were modified by the last command and

thus could be an interesting data-management problem for the systems designer. Nevertheless, the benefit to the user in having -- even just knowing of -the capability to withdraw a command could be quite important (e.g., easing the acute distress often experienced by new users, who are worried about 'doing something wrong').

The first two recovery situations (1 & 2) require the user to detect the undesired situation and initiate corrective actions. The second two situations (3 & 4) originate with the action of the system. For both of these kinds of situations it is most desirable to provide users with two pieces of information: what happened and why, and how to recover.

Recovery situation (3), abnormal terminations from program execution, from compilations, or from any program controlling the user session, typically occur because the program is being asked to do some proscribed action, and no code has been written to test for and handle that particular error. Often, in such cases, the user is given no more than an abrupt 'JOB ABORTED' or similar message. Nevertheless, the first part of the error message should simply indicate *what* rule was violated and *where* in the program this occurred. Secondly, users should be provided with information indicating what they could do to mend the situation, to continue processing. Ideally, perhaps, *all* system and user programs would run under the control of and be monitored by some supervisory system which somehow preserved, and helped restore, the world just prior to the abnormal termination.

Finally, even a meta-supervisor is of no help in recovery situation (4), when the system *crashes* -- experiences an unscheduled and abrupt termination of operation. Here, typically, all is lost. While data sets can be recovered in the form in which the user last remembered to save them before the crash, it would be more desirable to shift the burden of responsibility for the consequences of crashes onto the system. Thus, the system could provide for copying user files quite frequently onto mass storage media configured (and powered) independent of the

primary processor. Following a recovery from a crash, the system would automatically recreate the prior file environment.

1.2.8 On-Line Documentation

The user will require a variety of on-line information: about personal files (see section 1.2.3), about errors or problems encountered in using the facilities (see section 1.2.6), and, more generally, about what facilities are available and how to use them.

This notion of on-line documentation of system facilities is not a new one (e.g., Thompson, C., 1970), but development efforts are still largely experimental (e.g., HELP, 1976). The approach often suggested for and followed in interactive information retrieval systems is to have the user move down some hierarchical classification tree via choices from menus until an appropriate information document can be retrieved, typically on a key-word basis (e.g., Thompson, C., 1970; Thompson, D., 1971). This is a reasonable approach and can be very useful in saving users the problems of maintaining and searching among numbers of hard-copy manuals. One difficulty with this method is that the traversing of the hierarchy via the menus can be time-consuming and tedious.

A better solution would be to permit the user to invoke the assistance of the on-line facility via a natural language question. One demonstration of such an approach involves a modification of Weizenbaum's ELIZA program (Shapiro & Kwasny, 1975). The IBM Research HELP facility (1976) is a hybrid of the two approaches: after a user has invoked the facility, an attempt is made to respond to the user's natural language questions with a menu of pertinent choices. Because of the limitations of retrieval by KWIC (Key Word In Context) approaches, it would seem that further development of natural language assistance facilities will also have to be accompanied by automation of *conceptual* indexing of the reference manuals and other documents.

In addition to this work, which is directly concerned with providing users on-line information-retrieval assistance facilities, there is a great deal of promising research activity concerned with development of natural language systems for specialized applications (for an excellent review, see Walker, 1973; also see Malhotra, 1973; Malhotra & Sheridan, 1976; Thomas, 1976; Heidorn, 1976).

2. *INTERFACE CHARACTERISTICS*

2.1 DIALOGUE STYLE

We propose here a classification in which user-system dialogue styles can be viewed as differing basically in terms of two independent characteristics: (1) whether the interchange is guided overall by the user or the system, and (2) whether the user has to make a *choice* of his input from a set of alternatives presented by the system or else is able to provide a *free response*.

The party that is the 'guide' is the one that takes the initiative during the course of the exchange and also decides on a satisfactory termination point. For example, if a questionnaire were automated on a computer system, and various persons interacted with the system to provide their views, it would be the *system* that would be the guide. This is also true of some of the newer computerized sales check-out stations which prescribe a fixed routine for data-entry from the clerk. In most situations, however, the user is the guide.

If, in the above questionnaire situation, the system phrased questions in the form 'PLEASE ENTER YOUR AGE: ', the user would be unconstrained as to input and would be in the *free response* mode. However, if the question were phrased: 'PLEASE INDICATE YOUR AGE: (1) LESS THAN 18, (2) BETWEEN 18 AND 30, (3) OVER 30', the user would be in the *forced choice* mode. Although it might appear that when users select from the system's menu the system *must* be guiding the interaction, this forced-choice vs. free-response distinction is

independent of the question of who is the guide. For example, for a computer questionnaire, the computer creates a situation in which (1) the system is the guide, and (2) the user is in the forced-choice mode. However, in another situation the user may, for example, have invoked some on-line Help facility (see section 1.2.7) and is presented with a menu of selections from which to choose the closest to his information requirement. Here, the system is definitely *not* the guide, but the user is still in the forced-choice mode.

Since the two distinctions are independent of each other, there are four basic types of dialogues, and each has different advantages: *(1) System guides/user has forced-choice* -- this is most appropriate for routine task activities (see Introduction) where it is appropriate to step the user through a fixed procedure, providing menu selections at each step. Not only may this enhance the *speed* of accomplishing the task, but also, the restriction of input to a small set of alternatives greatly reduces the possibilities for *error*. When the user enters a number, e.g., item stock number, as part of the guided task, the forced-choice concept is still considered to apply, since (1) the user is forced to pick a number rather than anything else, and (2) quite often the number will be checked against expectations and rejected if out of bounds, etc. This guided/forced situation is also appropriate for structured interviews, surveys, or any information-gathering activity in which the topics and categories of interest can be specified ahead of time.

(2) System guides/user has free-response -- this dialogue is particularly appropriate when the overall objective is unstructured information-gathering -- such as an interview in which the user is asked general questions (e.g.,'What are your life goals'), where providing structure might actually interfere.

(3) User guides/user has forced-choice -- this combination is desirable when the user is at least somewhat knowledgeable about possible requests he could make of the system, or when the requirement to make a free-choice would be an unnecessary burden. This dialogue style is

often chosen for allowing a user to a select desirable system alternatives or information possibilities (see section 1.2.7).

(4) User guides/user has free-response -- this provides the maximum latitude for the user and is therefore most appropriate for experienced and confident users performing complex tasks. This situation also is the least structured and, with current technology, allows the maximum opportunity for miscommunication between user and system.

Heuristics for shifting from one to another of these modes can easily be imagined as a function of the user's difficulties and desires during the session. For example, in a user-guided interaction, the dialogue could be shifted from that of free-response to forced-choice either by signal of the user or on the basis of the system's detection of inordinate user difficulties.

The above classification is, of course, a great simplification, intended only as a very broad and general set of distinctions. When specific details of applications are taken into account -- e.g., how many alternatives, how information should be best arranged on the screen, whether a keyboard can be made available or not, etc. -there are a great number of possible variations. Martin (1973) would still appear to be the best reference source for guiding selection of any kind of special purpose dialogue style (also see Eason, Damodaran, & Stewart, 1975).

2.2 KEYBOARDS

2.2.1 Design of Keyboard Layouts

Delving into the question of the appropriate design of a full-character keyboard is a truly remarkable experience. Almost from the moment of its inception about a century past, one finds that the standard QWERTY keyboard has been under strong attack from a seemingly inexhaustible series of challengers, here claiming it has a poor arrangement relative to the letter sequences in English (e.g., Dvorak, Merrick, Dealey, & Ford, 1936), and there asserting it is the cause of a variety of aches and cramps (e.g., Ferguson & Duncan, 1974). In its place

have been proposed an inventor's-dream series of ingenious alternatives, many of these involving the idea of a 'chord', involving simultaneous depression of any number of keys (see review by Seibel, 1972). The fact remains, however, that *no* alternative has shown a realistically significant advantage over the QWERTY for general purpose typing (see the excellent review by Hanes, 1975).

2.2.2 Special Application Keyboards

When the typing application has special restrictions, then special purpose keyboards most probably must be devised -- as the chord-principle stenotype machine was designed to meet the speech-rate and silence requirements for courtroom and similar environments. In particular, when it is appropriate to restrict the alphabet in some way, , then the question of keyboard layout of the key subset is a very relevant performance consideration. The most obvious example is the layout of keyboards limited to numerical data entry. The two basic contenders involve a 3 x 3 arrangement of the digits 1-9 with 1-2-3 on the top (for touch-tone telephones) and 1-2-3 on the bottom (as in adding machines); a minor variation is whether the zero is placed above or below the matrix. While these variations produce comparable performance effects, marked deviations from them lead to degradations (e.g., Conrad, 1966; Conrad & Hull, 1968).

Aside from the very common numerical keyboards, there are an enormous number of special purpose keyboards with keys representing anything from phrases to graphic figures to whole programs. Here, careful analysis is required, and one should become familiar with the many issues underlying choice of such keyboards (see Martin, 1973; also, cf. Burch & Strater, 1974).

2.3 ALPHA-NUMERIC DISPLAYS

2.3.1 Physical Characteristics

There are a number of physical aspects to displays (both alpha-numeric *and* graphics) which affect perception and thus may impact performance -- e.g., flicker, luminance, CRT scanning patterns, contrast. Most of these factors are fairly well understood, and acceptable/desirable parameter values can be identified in most cases. Similarly, the effect of various character fonts on visibility and readability is also well known. Although one's choices may be restricted, it is nonetheless important to ensure that one's particular system is within preferred limits, and there are a number of excellent reviews which can be consulted (e.g., Gould, 1968; Martin, 1973; McLaughlin, 1973; Rouse, 1975).

2.3.2 Information Coding: Physical Variations

It is often important to discriminate among different classes of information simultaneously present on the display. One technique for accomplishing this involves varying the physical parameters of the characters. At least four options are possible, and each provides for a different capability of discriminating among a set of items (maximum recommended set size shown in parentheses; see Martin, 1973): *size* (e.g., as capitals vs. mixed cases, 2-4 items), *color* (6), *flashing* (2), and *brightness* (2).

2.3.3 Information Coding: Partitioning of the Display Screen

Independent of coding by character variations, the screen could be partitioned into separate areas for the following particular types of information: *(1) main work area* (e.g., 20 lines) -- either for text, system-supplied selection menus, or for a transaction record; *(2) input preparation area* (e.g., 1-2 lines) -- for generating (and locally editing) the next input to the system; *(3) system facility indicator* (e.g., 1/2 line) -- indicating what system facility has

been invoked (e.g., editor, compiler) and what the operating level of that facility is (e.g., for editors, the recursion level and tab settings, for compilers, the options and execution mode); *(4) diagnostic area* (e.g., 1 line) -- for indicating, when appropriate, what the nature of the condition was that terminated processing in some facility, and also what the user could do to restart; *(5) Fixed response area* (e.g., 1-4 lines) -- for implementations in which the dialogue (see section 2.1) is menu-driven or for applications in which a fixed set of inputs/responses are applicable for all activities within the application.

2.4 SPEECH INPUT/OUTPUT

2.4.1 Potential Levels of Processing

In the transmission of any set of meaningful symbols -- but particularly so in the case of speech input -- it is important to distinguish between three levels of processing which the receiver can perform. There is, first, what may be called *non-coded storage,* which involves only a reproduction and storage of the input -- corresponding to photostatic copies of documents in the visual domain, and to simple voice recordings in the audio domain. Secondly, there can be *recognition,* involving the (rule-governed) segmentation and coding of the input symbol stream into units which are 'meaningful' in the sense of possessing a matching pattern in some morphemic reference lexicon, which lexicon might also contain other information to be associated to the entries. Third, there is *understanding* or *comprehension* which implies a coding of the recognized lexical units for association with some *semantic* representation (e.g., a knowledge structure; see, e.g., Bobrow & Collins, 1975).

In the discussion of these levels which follows, the staggering difficulties of technically accomplishing the various kinds of processing are deliberately omitted. Since the technology for speech processing in most cases is unavailable except on a very limited research basis , the treatment here is necessarily more of a review of the state of the art than a discussion of, at present, indeterminable behavioral issues.

2.4.2 Speech Recordings

At the lowest level of symbol transmission and receipt -- non-coded storage -there is a remarkable amount of utility which can be realized in such a user-system interaction involving (computer-digitized) speech recording. Users can, first of all, use such a facility as a message distribution center to replace or augment document channels. Secondly, with suitable interpolation of signals to code sentence and paragraph, etc., structures, such a system could be used for later audio-editing (cf. section 1.2.2.3). Third, messages could be recorded and stored with certain classification codes which would permit their being accessed selectively (locally *and* remotely) as part of a general audio information storage and retrieval system. Finally, and of a different nature than the above, there is the potential of using speech recordings for automatically verifying the identity of the speaker (e.g., Das & Mohn, 1971).

2.4.3 Speech Recognition

Two levels of speech recognition can be identified: isolated word recognition and continuous speech recognition. Intermediate between these is a form of recognition of continuous speech into which have been inserted exaggerated inter-word delays by the speaker.

Almost perfect isolated word recognition for vocabularies of 30-200 words is, more or less, well within the state of the art (e.g., Flanagan, 1976; Hyde, 1972). Remembering that the lexicons for useful subsets of programming languages could be included within such a vocabulary, such a capability provides for at least technical (if not economic) feasibility in exercising considerable control over a computer system *by voice.*

Continuous speech recognition is a vastly more difficult task than single-word recognition, primarily because of the uncertainties involved in obtaining the correct word segmentations (e.g., Reddy, 1975). One of the obvious and interesting applications of such a processing capability is the idea of a speech typewriter: a user speaks into one end of the processor, and a

crisply typed manuscript appears at the other end (e.g., Flanagan, 1976; Jelinek, 1976). While a very narrowly-realized implementation of such a system might be able to be constructed in the next 10 years or so (e.g., Flanagan, 1976), any broader continuous speech recognition may well require a generation (and may well depend upon the progress in the following problem area).

2.4.4 Speech Understanding

Processing at this level is equivalent to that of natural language understanding. While extraordinary progress has been made in this field in the last decade, it is only now that the extent of the difficulties can be appreciated: understanding is not going to be very much a part of user-system interaction for many years (e.g., Bobrow & Collins, 1975; Elithorn & Jones, 1973; Heidorn, 1976; Reddy, 1975).

2.4.5 Speech Output

Computer-synthesized word production, on the other hand, is very much a here-now phenomenon (e.g., Flanagan, 1975; Smith & Goodwin, 1970). Electronic equipment can now output signals sounding remarkably like words produced by a human voice (male *or* female -- the latter being more difficult). However, natural sounding *continuous discourse* will require much additional research. From the point of view of the central computer processor, the speech synthesizer is just another peripheral output device, and thus almost anything that can be output on a typewriter could be spoken.

2.5 GRAPHICAL DEVICES

2.5.1 Display Devices Available

Two basic types of graphical displays are currently available: the standard cathode ray tube (CRT) and the more recent *storage devices*. The advantages of the former (aside from the

long experience with it) lie mainly in its selective write/erase capability. Its disadvantage is primarily that the image must constantly be refreshed, and this requires additional local storage buffering and refresh capability or else direct stimulation from the central computer processor.

The storage tube, on the other hand, requires the equivalent of only one CRT sweep to write the screen, and the image will persevere for an indefinite period of time. Although the refresh problem is therefore solved, storage tubes do not permit *selective* erasure at the present time -- it is all or nothing. A less serious problem of many storage tubes is the bright visually disconcerting flash that accompanies a screen erasure. Nevertheless, for many applications, the storage tube may be more desirable. (for a review of the display device considerations, see e.g., Walker, Gurd, & Drawneek, 1975; Newman & Sproull, 1974).

2.5.2 Input Devices for Graphics

Input devices can be partitioned into those that input information directly into the display device vs. those that input into a graphics tablet. The former category can further be divided into those devices which operate directly on the face of the display and those which remotely position the display's cursor.

Devices operating directly on the display face include the well-known light pen and, more recently, the user's pointing finger (triggering either acousticsensing or resistance-locating via a grid overlay). There is an immediacy to such devices that make them attractive, but there is the potential of disrupting keyboard entries by requiring such pointing or touching actions.

When substantial material must be input via the keyboard, it is probably preferable to use cursor-positioning devices which are closely associated with the keyboard (such as the four cursor keys on the IBM 3270 keyboard just to the right of the main QWERTY keyboard). Alternatively, some analogue device could be employed, such as a joystick, tracker-ball, or

something like Englebart's famous 'mouse' (see Prince, 1971; Ritchie & Turner, 1975; Walker, Gurd, & Drawneek, 1975).

The second major class of input devices, the tablets, involve a flat surface on which is drawn the desired material, which input is electronically sensed by several possible means, transmitted and re-created (with or without improvements such as straightening lines, etc.) on the primary display. These devices would appear to have their greatest utility either for free-hand irregular drawing or for tracing of other documents. (See Gammill, R.C., 1973; Corley & Allan, 1976; Ritchie & Turner, 1975; Walker, Gurd, & Drawneek, 1975).

2.5.3 Graphic Function

The functional capability that is required to support completely particular graphics applications has, it is argued, two components: a set of *specialized functional capability customized* for the particular application, this built *out of, or in addition to,* some general functional capability which is common to all graphical applications. It is this common base set of functions which we consider here. For examples of the additional customized functions, see, e.g.: for graphics and art, Csuri, 1974; for graphical circuit design, Walker, Gurd, & Drawneek, 1975, Chapter 1; for highway construction with graphic simulations, Moffett, 1974.

2.5.3.1 Graphical Primitives

It is presumed that in graphical applications, as in any application, it will be necessary to provide linguistic annotations at various points. Thus, some capability for alphanumeric input (and its editing) is required. The essential nature of *graphics,* however, involves the additional *capability to describe geometrical properties.* We view the graphical properties as being: (1) points, and (2) lines connecting the points.

Systems vary greatly in the manner in which these primitives, particularly lines, are defined. Indeed, the particular method of implementation is best dictated by the requirements of the application (as line drawing in one application, e.g., architecture, might best be accomplished by specifying the end points, whereas in another application, e.g., cartoon animation, the user might best specify the exact intermediate points). For discussions of primitive requirements, cf.: Foley & Wallace,1974; Newman & Sproull, 1974; Spence, 1976; Walker, Gurd, & Drawneek, 1975.

2.5.3.2 Data-Transformation Functions

Four types of transformations are required as primitive capability for graphics applications (e.g., Newman & Sproull, 1974; Walker, Gurd, & Drawneek, 1975).

SHIFTING: Performing an *additive* transformation of the (x,y, and possibly z) coordinates of all points of an object by some delta amount (e.g., $x' = x + k$). The result is a size-invariant spatial translation of the object.

SCALING: Performing a *multiplicative* transformation of the coordinates of an object by obtaining a product of some scaling factor, S (e.g., $x' = xS$ where the scaling factors on different axes need not be the same). The result is *compression* (for $S < 1$), *magnification* (for $S > 1$), or *reflection* about an axis (for negative-values S).

ROTATION: Performing a *trigonometric* transformation of object coordinates (e.g., $x' = x\cos A + y\sin A$ where A is the angle of rotation). The result is a *rotation* of the object about an intrinsic axis by angle A.

CLIPPING: Defining a rectangular area of the total viewing space such that only the object points which are within this space will be viewable.

WINDOWING: A derived transformation, but commonly referred to, involving, necessarily, a clipping transformation of the view, but, in addition, providing for other transformations simultaneously to be made.

2.5.3.3 Non-Manipulative Operator Functions

In addition to primitive object *transformations*, there are a basic set of non-manipulative operations to be applied to the graphical objects for which provision must be made (refer, also, to above references).

LOCATE: Provide for the identification of a particular point in the graphical space. This may be accomplished directly with one of the input devices which operate directly on the display face (e.g., light pen or finger, see section 2.5.2); alternatively, a cursor can be positioned with one of the indirect input devices such that it 'occupies' the desired point.

DEFINE OBJECT: Designate a located point -*and all points implied by that located point* -- to constitute some user-defined object. This object can subsequently be *named* (see below). The concept of *implied points* rests on some programmed basis for grouping or otherwise associating points in the graphical space. For example, the basis of association might be that of *continuity* of points. The defined object returned under this relation would be all of the points adjacently continuous to the located point. Thus, location of a point on a rectangle, under this principle, would return the whole rectangle as the user-defined object. (This is a modification of Folley & Wallace's concept).

NAME OBJECT: Assign some character-string *name* to a user-defined object. Multiple user-defined objects can thereby exist simultaneously, and operations and transformations can be performed on them by substituting the name as the argument instead of a point or other structure actually in the graphical space. This operation should also be extendable to permit the assignment of names to particular operations performed on objects.

VALUATE/EVALUATE OBJECT: The operation of *valuating* an object is the assignment of some *value* to that object with respect to some attribute or relation (e.g., length). This object-attribution act is distinct from *naming* in that the name applies to all aspects, including values, of an object, whereas a *value* is only one characteristic of an object.

The operation of *evaluation* returns the value assigned to an object on some particular attribute or relation.

3. A FINAL WORD

Although we have touched on many issues involved in many aspects of user-computer interaction, undoubtedly there are numerous issues we have *not* broached, or others we have dealt with only superficially or with too strong a bias. Such is the characteristic of a complex field -- that no one review does justice to the whole area-- and it is most encouraging to us that concern with the *behavioral* issues of computer systems now is growing full-size to equal status with the more traditional concerns with computer architecture and internal software.

In view of the expected incompleteness of our treatment here, we very much welcome additions and critiques from you, the reader, with the intent of updating this review with your contributions at some time in the future.

REFERENCES

Adams, J. and Cohen, E. Time-sharing vs. instant batch processing: An experiment in programming training. *Computers and Automation*, March 1969, 30-34.

Allen, F.E. Interprocedural data flow analysis. *IFIPS Congress report*, 398-402, 1974.

Allen, F.E. and Cocke, J. A program data flow analysis procedure *IBM Research Report*, RC 5257, 1975.

Aron, J.D. *The program development process: Part I. The individual programmer*. Reading, Mass: Addison-Wesley Publishing Co., 1974.

Balzer, R.M. EXDAMS -- EXtendable Debugging And Monitoring System. *AFIPS Conference Proceedings*, Volume 34, 1969.

Bennett, J. L. The user interface in interactive systems. In Cuadra, C.A. (Ed.) *Annual Review of Information Science and Technology*. Volume 7. Washington, D.C.: American Society for Information Science, 1972, 159-196.

Bennett, J.L. User acceptance of decision support systems: The role of an integrating agent. *IBM Research Report*, RJ 1502, 1975.

Bobrow, D.G. and Collins, A. (Eds.) *Representation and Understanding*. New York: Academic Press, Inc., 1975.

Boehm, B.W., Seven, M.J., and Watson, R.A. Interactive problem-solving -An experimental study of 'lockout' effects. *Proceedings of Spring Joint Computer Conference*, 1971, 205-210.

Boies, S.J. User behavior in an interactive computer system. *IBM Systems Journal*, 1974, *13*, 1-18.

Boies, S.J. and Gould, J.D. Syntactic errors in computer programming. *Human Factors*, 1974, *16*, 253-257.

Boies, S.J. and Spiegel, M.F. A behavioral analysis of programming: On the use of interactive debugging facilities. *IBM Research Report*, RC 4472, 1973.

Brown, A.R. and Sampson, W.A. *Program Debugging*. New York: American Elsevier, 1973.

Burch, J.G., Jr., and Strater, F.R., Jr. *Information Systems: Theory and Practice*. Santa Barbara, California: Hamilton Publishing Co., 1974.

Burton, R.R. A semantically centered parsing system for mixed-initiative CAI systems. Paper presented at the Association for Computational Linguistics Conference, Amherst, Mass., July, 1974.

Carbonell, J. R. On man-computer interaction: A model and some related issues. *IEEE Trans. on System Science and Cybernetics*, 1969, SSC-5.

Carbonell, J.R., Elkind,J.I., and Nickerson, R.S. On the psychological importance of time in a time-sharing system. *Human Factors*, 1968, *10*, 135-142.

Carlson, E.D. Using large data bases for interactive problem solving. *IBM Research Report*, RJ 1585, 1975.

Chamberlin, D. D. and Boyce, R. F. SEQUEL---A Structured English QUery Language. *IBM Research Report*, RJ 1394, 1974.

Chapanis, A. Prelude to 2001: explorations in human communication. *American Psychologist*, 1971, *26*, 949-961.

Codd, E. F. Seven steps to rendezvous with the casual user. *IBM Research Report*, RJ 1333, 1974.

Conrad, R. Short-term memory factor in the design of data-entry keyboards. *Journal of Applied Psychology*, 1966, *50*, 353-356.

Conrad, R. and Hull, A.J. The preferred layout for numeral data-entry keysets. *Ergonomics*, 1968, *11*, 165-73.

Corley, M.R. and Allan, J.J. Pragmatic information processing aspects of graphically accessed computer-aided design. *IEEE Transactions on Systems, Man, and Cybernetics*, 1976, *SMC-6, 434-439*.

Csuri, C. Computer graphics and art. *Proceedings of the IEEE*, 1974, *62*, 503-515.

Das, S.K. and Mohn, W.S. A scheme for speech processing in automatic speaker verification. *IEEE Transactions on Audio and Electroacoustics*, 1971, *AU-19*, 32-43.

Davis, R.M. Man-machine communication. In Cuadra, C.A. (Ed.) *Annual Review of Information Science and Technology*. Volume 1. New York: Interscience Publishers, 1966, 221-254.

Deutsch, L.P. and Lampson, B.W. An online editor. *Communications of the ACM*, 1967, *10*, 793-799.

Doherty, W.J., Thompson, C.H., and Boies, S.J. An analysis of interactive system usage with respect to software, linguistic, and scheduling attributes. *IBM Research Report*, RC 3914, 1972.

Donzeau-Gouge, V., Huet, G., Kahn, G., Lang, B., and Levy, J.J. A structure oriented program editor: a first step towards computer assisted programming. *Laboratoire de recherche en informatique et automatique Rapport de Recherche*, No. 114, 1975.

Dunn, D.A. Communications technology. In Cuadra, C.A. and Luke, A.W. (Eds.) *Annual Review of Information Science and Technology*. Washington, D.C.: American Society For Information Science, 1975, 165-193.

Dvorak, A., Merrick, N.L., Dealey, W.L., and Ford, G.C. *Typewriting Behavior.* New York: American Book Co., 1936.

Eason, K.D., Damodaran, L., and Stewart, T.F.M. Interface problems in man-computer interaction. In Mumford, E. and Sackman, H. (Eds.) *Human Choice and Computers.* New York: American Elsevier Publishing Co., 1975.

Elithorn, A. and Jones, D. (Eds.) *Artificial and Human Thinking.* San Francisco: Jossey-Bass Inc., 1973

Elson, M. *Concepts of programming languages.* Chicago: Science Research Associates, Inc., 1973.

Engel, S.E. and Granda, R. E. Guidelines for man/display interfaces. *IBM Technical Report,* TR 00.2720, 1975.

Englebart, D.C. Augmenting human intellect: A conceptual framework. *Stanford Research Institute Report,* Contract AF 49(638)-1024 (AD 289 565), 1962.

Englebart, D.C., Watson, R.W., and Norton, J.C. The augmented knowledge workshop. In *AFIPS National Computer Conference Proceedings.* Volume 42. Montvale, N.J.: AFIPS Press, 1973, 9-21.

Fajman, R. and Borgelt, J. WYLBUR: An interactive text editing and remote job entry system. *Communications of the ACM,* 1973, *16,* 314-322.

Ferguson, D. and Duncan, J. Keyboard design and operating posture. *Ergonomics,* 1974, *17,* 731-744.

Flanagan, J.L. Computers that talk and listen: man-machine communication by voice. *Proceedings of the IEEE,* 1976, *64,* 405-415.

Foley, J.D. and Wallace, V.I. The art of natural graphic man-machine conversation *Proceedings of the IEEE,* 1974, *62,* 462-471.

Gaines, R.S. The debugging of computer programs. *Institute for Defense Analysis Report,* Princeton, N.J., 1969.

Gammill, R.C. Graphics and interactive systems -- design considerations of a software system. *National Computer Conference Proceedings,* 1973.

Goldberg, P.C. Structured programming for non-programmers. *IBM Research Report,* RC 5318, 1975.

Gorry, G.A. and Morton, M.S. A framework for management information systems. *Sloan Management Review,* 1971, *12,* 55-70.

Gould, J.D. Visual factors in the design of computer-controlled CRT displays. *Human Factors,* 1968, *10,* 359-376.

Grace, B.F. A case study of man/computer problem-solving. *IBM Research Report,* RJ 1483, 1975.

Hammer, M., Howe, W.G., Kruskal, V.J., and Wladawsky, I. A very high level programming language for data processing applications. *IBM Research Report*, RC 5583, 1975.

Hanes, L.F. Human factors in international keyboard arrangement. In Chapanis, A. (Ed.) *Ethnic Variables in Human Factors Engineering*. Baltimore, Md.: The Johns Hopkins University Press, 1975, 189-206.

Hantler, S.L. and King, J.C. An introduction to proving the correctness of programs. *IBM Research Report*, RC 5893, 1976.

Harrison, W. Compiler analysis of the value ranges for variables. *IBM Research Report*, RC 5544, 1975.

Heidorn, G. Automatic programming through natural language dialogue: A survey. *IBM Journal of Research and Development*, 1976, *20*, 302-313.

HELP, the on-line VM168 user assistance facility, IBM Watson Research Center, developed by C. Thompson, (available only on-line via the system), 1976.

Hyde, S.R. Automatic speech recognition: a critical survey and discussion of the literature. In David, E.E., Jr. and Denes, P.B. (Eds.) *Human Communication: A Unified View*. New York: McGraw-Hill Book Co., 1972.

Jelinek, F. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*. 1976, *64*, 532-556.

Kelly, M.J. Studies in interactive communication: limited vocabulary natural language dialogue. *Johns Hopkins University Department of Psychology Technical Report*, No. 3, 1975.

IBM TERMTEXT/Format Language Guide. Reference Manual SH20-1372-0, Program Number 5796-PBR, 1973.

IBM SCRIPT/370 Program Description and Operations Manual. Reference Manual SH20-1114, 1975.

IBM Virtual Machine Facility/370: CMS User's Guide. IBM Reference Manual GC20-1819-0, 1976.

Kemeny, J.G. *Man and the Computer*. New York: Charles Scribner and Sons, 1972.

Kennedy, T.C.S. The design of interactive procedures for man-machine communication. *International Journal of Man-Machine Studies*, 1974, *6*, 309-334.

King, J.C. Symbolic execution and program testing. *IBM Research Report*, RC 5082, 1974.

Kruskal, V.J. An editor for parametric programs. *IBM Research Report*, RC 6070, 1976.

Licklider, J.C. Man-computer communication. In Cuadra, C.A.(Ed.) *Annual Review of Information Science and Technology*. Volume 3. Chicago: Encyclopedia Britannica, 1968, 201-240.

Malhotra, A. Design criteria for a knowledge-based english language system for management: an experimental analysis. *M.I.T. Project MAC Report*, TR-146, 1975.

Malhotra, A. and Sheridan, P.B. Experimental determination of design requirements for a program explanation system. *IBM Research Report*, RC 5831, 1976.

Mann, W.C. Dialogue-based research in man-machine communication. *USC/Information Sciences Institute Report*, ISI/RR-75-41, 1975.

Mann, W.C., Moore, J.A., Levin, J.A., Carlisle, J. H. Observation methods for human dialogue, *USC/Information Sciences Institute Report*, ISI/RR-75-33, 1975.

Martin, J. *Design of Man-Computer Dialogues*. Englewood Cliffs, N.J.: Prentice-Hall, 1973.

Martin, T.H. The user interface in interactive systems. In Cuadra, C.A. (Ed.) *Annual Review of Information Science and Technology*, Volume 8. Washington, D.C.: American Society for Information Science, 1973.

Martin, T.H., Carlisle, J.H., and Treu, S. The user interface for interactive bibliographic searching: an analysis of the attitudes of nineteen information scientists. *Journal of the American Society for Information Science*, 1973, *24*, 142-147.

McCormick, E. J. *Human factors engineering*. New York: McGraw-Hill, 1970.

McDonald, N. Getting started in INGRES with CUPID --- a tutorial. *Electronics Research Laboratory Memorandum*, ERL-M546, 1975a.

McDonald, N. CUPID --- A graphics oriented facility for support of non-programmer interactions with a data base. *Electronics Research Laboratory Memorandum*, ERL-M563, 1975b.

McLaughlin, R.A. Alphanumeric display terminal survey. *Datamation*, November, 1973, 71-92.

Meadow, C. T. *Man-machine communication*. New York: John Wiley and Son, 1970.

Meister, D. and Rabideau, G. F. *Human factors evaluation in system development*. New York: John Wiley and Sons, 1965.

Metzger, P.W. *Managing a programming project*. Englewood Cliffs, N.J.: Prentice-Hall, 1973.

Miller, L.A. Harlan Mills on 'The Psychology of Quality'. *IBM Research Report*, RC 3779, 1973.

Miller, L.A. Programming by non-programmers. *International Journal of Man-Machine Studies*, 1974, *6*, 237-260.

Miller, L.A. Naive programmer problems with specification of transfer-of-control. *AFIPS National Computer Conference*, Anaheim, Volume 44, 1975.

Miller, L.A. Natural language procedures: Guides for programming language design. Paper presented at the International Ergonomics Association Meeting, University of Maryland, 1976.

Miller, L.A. and Thomas, J.C., Jr. A functional taxonomy of problem-solving activities. In preparation.

Miller, R.B. Response times in computer conversational transactions. *Proceedings of Fall Joint Computer Conference*, 1968.

Miller, R.B. Archetypes in man-computer problem solving. *Ergonomics*, 1969, *12*, 559-581.

Miller, R.B. Human ease of use criteria and their tradeoffs. *IBM Technical Report*, TR 00.2185, 1971.

Mittman, B. and Borman, L. *Personalized Data Base Systems*. Los Angeles: Melville Publishing Co., 1975.

Moffett, T.J. Building highway systems with computer graphic simulations. *Proceedings of the IEEE*, 1974, *62*, 429-436.

Moulton, P.G. and Muller, M.E. DITRAN -- a compiler emphasizing diagnostics. *Communications of the ACM*, 1967, *10*, 45-52.

Nagy, G. and Pennebaker, M. A step toward automatic analysis of logically *IBM Research Report*, RC 3407, 1971.

Newman, W.M. and Sproull, R.F. An approach to graphics system design. *Proceedings of the IEEE*, 1974, *62*, 471-483.

Newsted, P.R. and Wynne, B.E. Augmenting man's judgment with interactive computer systems. *International Journal of Man-Machine Studies*, 1976, *8*, 29-59.

Ochsman, R.B. and Chapanis, A. The effects of 10 communication modes on the behavior of teams during co-operative problem-solving. *International Journal of Man-Machine Studies*, 1974, *6*, 579-619.

Palme, J. Interactive software for humans. *Swedish National Defense Research Institute, Planning and Operations Research FOA 1 Report*, No. C10029-M3(E5), 1975.

Pratt, T.W. *Programming languages: Design and implementation*. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

Prince, M.D. *Interactive Graphics for Computer-Aided Design*. Reading, Mass.: Addison-Wesley, 1971.

Reaser, J.M., Priesman, I., and Gill, J.F. A production environment evaluation of interactive programming. *U.S. Army Computer Systems Command*, Technical Documentary Report, USACSC-AT-74-03, 1974.

Reddy, D.R. *Speech Recognition*. New York: Academic Press, 1975.

Rescher, N. *The logic of commands.* New York: Dover, 1966.

Ritchie, G.J. and Turner, J.A. Input devices for interactive graphics. *International Journal of Man-Machine Studies,* 1975, *7,* 639-660.

Rosen, B.K. High level data flow analysis. Part 2 (escapes and jumps). *IBM Research Report,* RC 5744, 1975.

Rouse, W.B. Design of man-computer interfaces for on-line interactive systems. *Proceedings of the IEEE,* 1975, *63,* 847-857.

Rustin, R. (Ed.) *Debugging Techniques in Large Systems.* Englewood Cliffs, N.J.: Prentice-Hall, 1971.

Sackman, H. *Man-computer problem solving.* Princeton, N.J.: Auerbach, 1970.

Sammet, J.E. *Programming languages: History and fundamentals.* Englewood Cliffs, N.J.: Prentice-Hall, 1969.

Satterthwaite, E. Debugging tools for high level languages. *Software-practice and experience,* 1972, *2,*

Sauvain, R.W. Structural communication in a personal information storage and retrieval system. *University Microfilms,* Ann Arbor, Michigan, No. 70-21 782, 1971.

Schoonard, J.W. and Boies, S.J. Short-type: a behavioral analysis of typing and text entry. *IBM Research Report,* RC 4434, 1973.

Schwartz, M.C. Operational characteristics in a time-sharing environment. *IBM Technical Report,* TR 27.031, 1969.

Seibel, R. Data entry devices and procedures. In Van Cott, H.P. and Kincade, R. G. (Eds.) *Human engineering guide to equipment design,* Washington, D.C.: U.S. Government Printing Office, 1972.

Shackel, B. Man-computer interaction -- The contribution of the human sciences. *Ergonomics,* 1969, *12,* 485-499.

Shapiro, S. and Kwasny, S. Interactive consulting via natural language. *Communications of the ACM,* August, 1975.

Sime, M.E., Green, T.R.G., and Guest, D.J. Psychological evaluation of two conditional constructions used in computer languages. *International Journal of Man-Machine Studies,* 1973, *5,* 105-113.

Sime, M.E. and Green, T.R.G. Psychology and the syntax of programming. *Medical Research Council Social and Applied Psychology Unit Memo,* No. 52, 1974.

Singleton, W.T. *Man-machine systems.* Baltimore, Md.: Penguin Education, 1974.

Skinner, V.L., Jr. Text processing applications. *IBM Technical Report,* Federal System Division FSC 72-6014, 1972.

Smith, S.L. and Goodwin, N.C. Computer-generated speech and man-computer interaction. *Human Factors,* 1970, *12,* 215-223.

Spence, R. Human factors in interactive graphics. *Computer Aided Design,* 1976, *8,* 49-53.

Stone & Webster Engineering. Reported on in Engineering, editing, and problem solving. *IBM Computing Report,* Fall, 1973, 8-9.

Tapscott, R.P. ADS: The source listing annotator. *IBM Research Report,* RC 5065, 1974.

Tarnawsky, G.O. User semantic elements. *IBM ASDD Technical Report,* 1972.

Thomas, J. C. Quantifiers and question-asking. *IBM Research Report,* RC 5866, 1976a.

Thomas, J. C. A method for studying natural language dialogue. *IBM Research Report,* RC 5882, 1976b.

Thomas, J. C. and Gould, J. D. A psychological study of Query By Example. *IBM Research Report,* RC 5124, 1974.

Thompson, C.H. An information facility for the on-line user. *SHARE XXXIV Proceedings,* March, 1970.

Thompson, D.A. Interface design for an interactive retrieval system: a literature survey and a research system description. *Journal of the American Society for Information Science,* 1971, 361-373.

Tomeski, E. A. and Lazarus, H. *People-oriented computer systems: The computer in crisis.* New York: Van Nostrand Reinhold, 1975.

Unger, C. (Ed.) *Command languages.* (Proceeding of the IFIP working conference on command languages). New York: American Elsevier, 1975.

Van Dam, A. and Rice, D.E. On-line text editing: a survey. *Computing Surveys,* 1971, *3,* 93-114.

Van Cott, H. P. and Kinkade, R. G. (Eds.) *Human engineering guide to equipment design.* Washington, D.C.: U.S. Government Printing Office, 1972.

Van Tassel, D. *Program style, design, efficiency, debugging, and testing.* Englewood Cliffs, N.J.: Prentice-Hall, 1974.

Walker, B.S., Gurd, J.R., and Drawneek, E.A. *Interactive Computer Graphics.* New York: Crane, Russak & Co., 1975.

Walker, D.E. (Ed.) *Interactive Bibliographic Search: The User/Computer Interface.* Montvale, New Jersey: AFIPS Press, 1971.

Walker, D.E. Automated language processing. In Cuadra, C.A. and Luke, A.W. (Eds.) *Annual Review of Information Science and Technology. Volume 8.* Washington, D.C.: American Society for Information Science, 1973.

Walter, E.S. and Wallace, V.L. Further analysis of a computing center environment. *Communications of the ACM*, 1967, *10*, 266-272. Weinberg, G.M. *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold, 1971.

Wilks, Y. Semantic considerations in text processing. In Gloye, E.E. and Marcus, R.J. (Eds.) *Computer text processing and scientific research*. Conference proceedings. Pasadena, California, March, 1973, 39-54.

Wright, P. and Fox, K. Presenting information in tables. *Applied Ergonomics*, 1970, *1*, (4), 234-242.

Yntema, D.B. The Cambridge Project: Computer methods for analysis and modeling of complex systems. *Rome Air Development Technical Report*, RADC-TR-74-159, 1974.

Zloof, M. M. Query By Example. *IBM Research Report*, RC 4917, 1974.

Zloof, M. M. Query By Example: the invocation and definition of tables and forms. *IBM Research Report*, RC 5115, 1975.

TECHNICAL REPORTS DISTRIBUTION LIST

CODE 455

Director, Engineering Psychology (5 cys)
Programs, Code 455
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217

Defense Documentation Center (12 cys)
Cameron Station
Alexandria, Virginia 22314

Director, ONR Branch Office
ATTN: Dr. J. Lester
495 Summer Street
Boston, Massachusetts 02210

Director, ONR Branch Office
ATTN: Dr. M. Bertin
536 S. Clark Street
Chicago, Illinois 60605

Director, ONR Branch Office
ATTN: Dr. E. Gloye
1030 East Green Street
Pasadena, California 91106

Director, ONR Branch Office
ATTN: Mr. R. Lawson
1030 East Green Street
Pasadena, California 91106

Dir., Naval Research Laboratory (6 cys)
Technical Information Division
Code 2027
Washington, D. C. 20375

Dir., Naval Research Laboratory (6 cys)
ATTN: Library, Code 2029 (ONRL)
Washington, D. C. 20375

Mr. John Hill
Naval Research Laboratory
Code 5634
Washington, D. C. 20375

Office of Naval Research
Information Systems Program, Code 437
800 North Quincy Street
Arlington, Virginia 22217

Office of Naval Research
Operations Research Program, Code 434
800 North Quincy Street
Arlington, Virginia 22217

Office of Naval Research
Naval Analysis Programs, Code 431
800 North Quincy Street
Arlington, Virginia 22217

Office of Naval Research
ATTN: Dr. K. T. Wallenius, Code 436
800 North Quincy Street
Arlington, Virginia 22217

Office of the Chief of Naval
Operations, Op-987E
Department of the Navy
Washington, D. C. 20350

CDR H. J. Connery
Office of the Chief of Naval Operations,
Op-987M4
Department of the Navy
Washington, D. C. 20350

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code AX
Washington, D. C. 20380

Dr. Heber G. Moore
Hqs. Naval Material Command
Code 0331
Department of the Navy
Washington, D. C 20360

Mr. Arnold Rubinstein
Naval Material Command, NAVMAT 03424
Department of the Navy
Washington, D. C. 20360

Commander, Naval Electronics
Systems Command
Command and Control Div., Code 530
Washington, D. C. 20360

Naval Electronics Systems Command
Human Factors Engineering Branch
Code 4701
Washington, D. C. 20360

Bureau of Medicine and Surgery
Human Effectiveness Branch, Code 713
Department of the Navy
Washington, D. C. 20372

CDR Robert Wherry
Human Factors Engineering Branch
Crew Systems Department
Naval Air Development Center
Johnsville
Warminster, Pennsylvania 18974

LCDR Robert Kennedy
Human Factors Engineering Br., Code 5342
U.S. Naval Missile Center
Point Mugu, California 93042

Lt. Col., Henry L. Taylor, USAF
OAD*E&LS) ODDR&E
Pentagon, Rm. 3D129
Washington, D. C. 20301

Mr. Richard Coburn
Head, Human Factors Division
Naval Electronics Laboratory Center
San Diego, California 92152

Dean of Research Administration
Naval Postgraduate School
Monterey, California 93940

Navy Personnel Research and
Development Center (Code 10) (5 cys)
San Diego, California 92152

Mr. James L. Long
Weapons Systems Research (N-332)
Naval Education and Training Command
Naval Air Station
Pensacola, Florida 32408

Human Factors Dept., Code N215
Naval Training Equipment Center
Orlando, Florida 32813

Dr. George Moeller
Head, Human Factors Engineering Branch
Submarine Medical Research Laboratory
Naval Submarine Base
Groton, Connecticut 06340

Lt. Col. Austin W. Kibler
Director, Human Resources Office
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, Virginia 22209

U.S. Air Force Office of Scientific
Research
Life Sciences Directorate, NL
1400 Wilson Blvd.
Arlington, Virginia 22209

Dr. J. M. Cristensen
Chief, Human Engineering Division
Aerospace Medical Research Laboratory
Wright-Patterson AFB, OH 45433

Dr. J. E. Uhlaner
Dir., U.S. Army Research Institute
for the Social & Behavioral Sciences
1300 Wilson Blvd.
Arlington, Virginia 22209

Chief of Research and Development
Human Factors Branch
Behavioral Science Division
Department of the Army
ATTN: Mr. J. Barber
Washington, D. C. 20310

Dr. Joseph Zeidner
Dir., Organization and Systems
Research Laboratory
U.S. Army Research Institute for
the Behavioral & Social Sciences
1300 Wilson Blvd.
Arlington, Virginia 22209

Dr. Stanley Deutsch
Chief, Man-Systems Integration
OART, Hqs., NASA
600 Independence Avenue
Washington, D. C. 20546

Dr. Jesse Orlansky
Institute for Defense Analyses
400 Army-Navy Drive
Arlington, Virginia 22202

Dr. Edgar M. Johnson
Organizations & Systems Research Lab.
U.S. Army Research Institute for the
Behavioral and Social Sciences
1300 Wilson Blvd.
Arlington, Virginia 22209

Dr. James Parker
BioTechnology, Inc.
3027 Rosemary Lane
Falls Church, Virginia 22042

Dr. Edwin A. Fleishman
Foxhall Square
3301 New Mexico Avenue, N.W.
Washington, D. C. 20016

American Institutes for Research Library
135 N. Bellefield Avenue
Pittsburgh, Pennsylvania 15213

Psychological Abstracts
American Psychological Association
1200 17th Street, N.W.
Washington, D. C. 20036

Dr. A. I. Siegal
Applied Psychological Services
404 E. Lancaster Street
Wayne, Pennsylvania 19087

Dr. Joseph Wulfeck
Dunlap and Associates, Inc.
115 South Oak Street
Inglewood, California 90301

Dr. Robert R. Mackie
Human Factors Research, Inc.
Santa Barbara Research Park
6780 Cortona Drive
Goleta, California 93017

Mr. Wes Woodson
Man Factors, Inc.
4433 Convoy Street, Suite D
San Diego, California 92111

Dr. C. H. Baker
Director, Human Factors Wing
Defense & Civil Institute of
Environmental Medicine
P. O. Box 2000
Downsville, Toronto, Ontario
Canada

Journal Supplement Abstract Service
American Psychological Association
1200 17th Street, N.W.
Washington, D. C. 20036

Dr. Bruce M. Ross
Department of Psychology
Catholic University
Washington, D. C. 20017

Mr. Harry Chipman
WR Systems, Inc.
2531 S. Jefferson Davis Highway
Arlington, Virginia 22202

Dr. David Meister
U.S. Army Research Institute
1300 Wilson Blvd.
Arlington, Virginia 22209

Mr. George Graine
Naval Ship Systems Command
(SHIPS 047C12)
Department of the Navy
Washington, D. C. 20362

Lt. Col. John F. Ahlborn
Headquarters, AFSC-DLSE
Andrews AFB
Washington, D. C. 20334

Dr. H. H. Wolff
Technical Director (Code N-2)
Naval Training Equipment Center
Orlando, Florida 32813

Dr. Donald A. Topmiller
Chief, Systems Effect. Branch
Human Engineering Division, USAF
Wright Patterson AFB, Ohio 45433

Col. Robert O. Viterna
DA/OCRD
Headquarters, Dept. of the Army
DARD-ARS-B
Washington, C. C. 20310

Dr. Anthony Debons
IDIS
University of Pittsburgh
135 N. Bellefield Avenue
Pittsburgh, Pennsylvania 15260

Dr. Alfred F. Smode
Training Analysis & Evaluation Group
Code N-OOT
Orlando, Florida 32813